Constructor Chaining In Java with Examples

Difficulty Level : Easy • Last Updated : 04 Jul, 2022

Constructor chaining is the process of calling one constructor from another constructor with respect to current object.

One of the main use of constructor chaining is to avoid duplicate codes while having multiple constructor (by means of constructor overloading) and make code more readable.

Prerequisite - Constructors in Java

Constructor chaining can be done in two ways:

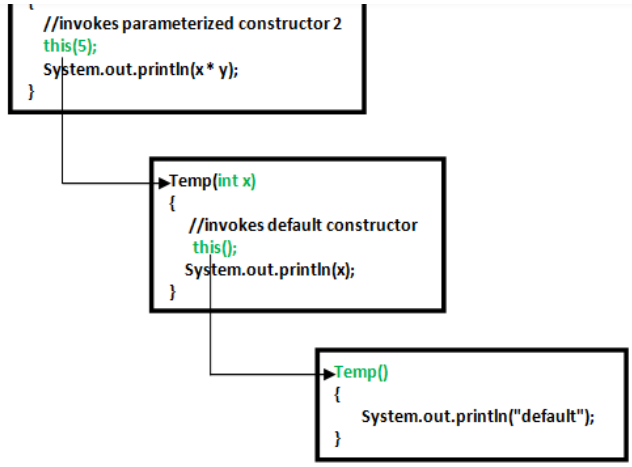
- **Within same class**: It can be done using **this()** keyword for constructors in the same class
- **From base class:** by using **super()** keyword to call the constructor from the base class.

Constructor chaining occurs through **inheritance**. A sub-class constructor's task is to call super class's constructor first. This ensures that the creation of sub class's object starts with the initialization of the data members of the superclass. There could be any number of classes in the inheritance chain. Every constructor calls up the chain till the class at the top is reached.

Why do we need constructor chaining?

This process is used when we want to perform multiple tasks in a single constructor rather than creating a code for each task in a single constructor we create a separate constructor for each task and make their chain which makes the program more readable.

Constructor Chaining within the same class using this() keyword:



Java

```java
// Java program to illustrate Constructor Chaining
// within same class Using this() keyword
class Temp
{
    // default constructor 1
    // default constructor will call another constructor
    // using this keyword from same class
    Temp()
    {
        // calls constructor 2
        this(5);
        System.out.println("The Default constructor");
    }

    // parameterized constructor 2
    Temp(int x)
    {
        // calls constructor 3
        this(5, 15);
        System.out.println(x);
    }

    // parameterized constructor 3
    Temp(int x, int y)
    {
        System.out.println(x * y);
    }

    public static void main(String args[])
    {
        // invokes default constructor first
        new Temp();
    }
```

Start Your Coding Journey Now!

Output:

```
75
5
The Default constructor
```

Rules of constructor chaining :

1. The **this()** expression should always be the first line of the constructor.
2. There should be at-least be one constructor without the this() keyword (constructor 3 in above example).
3. Constructor chaining can be achieved in any order.

What happens if we change the order of constructors?

Nothing, Constructor chaining can be achieved in any order

Java

```
// Java program to illustrate Constructor Chaining
// within same class Using this() keyword
// and changing order of constructors
class Temp
{
    // default constructor 1
    Temp()
    {
        System.out.println("default");
    }

    // parameterized constructor 2
    Temp(int x)
    {
        // invokes default constructor
        this();
        System.out.println(x);
    }

    // parameterized constructor 3
    Temp(int x, int y)
    {
        // invokes parameterized constructor 2
        this(5);
        System.out.println(x * y);
```

```
    {
        // invokes parameterized constructor 3
        new Temp(8, 10);
    }
}
```

Output:

```
default
5
80
```

NOTE: In example 1, default constructor is invoked at the end, but in example 2 default constructor is invoked at first. Hence, order in constructor chaining is not important.

Constructor Chaining to other class using super() keyword :

Java

```
// Java program to illustrate Constructor Chaining to
// other class using super() keyword
class Base
{
    String name;

    // constructor 1
    Base()
    {
        this("");
        System.out.println("No-argument constructor of" +
                                            " base class");
```

```
    Base(String name)
    {
        this.name = name;
        System.out.println("Calling parameterized constructor"
                                                + " of base");
    }
}

class Derived extends Base
{
    // constructor 3
    Derived()
    {
        System.out.println("No-argument constructor " +
                           "of derived");
    }

    // parameterized constructor 4
    Derived(String name)
    {
        // invokes base class constructor 2
        super(name);
        System.out.println("Calling parameterized " +
                           "constructor of derived");
    }

    public static void main(String args[])
    {
        // calls parameterized constructor 4
        Derived obj = new Derived("test");

        // Calls No-argument constructor
        // Derived obj = new Derived();
    }
}
```

Output:

```
Calling parameterized constructor of base
Calling parameterized constructor of derived
```

Note : Similar to constructor chaining in same class, **super()** should be the first line of the constructor as super class's constructor are invoked before the sub class's constructor.

Alternative method : using Init block :

constructor, whenever a constructor is used for creating a new object.

Example 1:

Java

```
class Temp
{
    // block to be executed before any constructor.
    {
        System.out.println("init block");
    }

    // no-arg constructor
    Temp()
    {
        System.out.println("default");
    }

    // constructor with one argument.
    Temp(int x)
    {
        System.out.println(x);
    }

    public static void main(String[] args)
    {
        // Object creation by calling no-argument
        // constructor.
        new Temp();

        // Object creation by calling parameterized
        // constructor with one parameter.
        new Temp(10);
    }
}
```

Output:

```
init block
default
init block
10
```

Example :

Java

```
class Temp
{
    // block to be executed first
    {
        System.out.println("init");
    }
    Temp()
    {
        System.out.println("default");
    }
    Temp(int x)
    {
        System.out.println(x);
    }

    // block to be executed after the first block
    // which has been defined above.
    {
        System.out.println("second");
    }
    public static void main(String args[])
    {
        new Temp();
        new Temp(10);
    }
}
```

Output :

```
init
second
default
init
second
10
```

Start Your Coding Journey Now!

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Related Articles

1. Java Program to Show Inherited Constructor Calls Parent Constructor By Default
2. Method Chaining In Java with Examples
3. Implementing our Own Hash Table with Separate Chaining in Java
4. Java Program to Implement Hash Table Chaining with List Heads
5. Java Program to Implement Hash Tables Chaining with Doubly Linked Lists
6. Constructor getAnnotatedReturnType() method in Java with Examples
7. Constructor getAnnotatedReceiverType() method in Java with Examples
8. Constructor equals() method in Java with Examples
9. Constructor getDeclaringClass() method in Java with Examples
10. Constructor getName() method in Java with Examples

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Easy

Easy | Normal | Medium | Hard | Expert

Improved By : vkbvipul, Vikramaditya Kukreja, nidhi_biet, anikakapoor, kalrap615, jeevanmali1994

Article Tags : Java, School Programming

Practice Tags : Java

Improve Article | Report Issue

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Start Your Coding Journey Now!